Modelling and validating distributed systems with TLA+

Carla Ferreira
29th April 2019
TLA+ specification language

- Formal language for describing and reasoning about distributed and concurrent systems.

- TLA+ is a model-oriented language:
  - based on mathematical logic and set theory plus temporal logic TLA (temporal logic of actions).

- Supported by the TLA Toolbox.

- References:
Turing Award 2013

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.
Use of TLA+ at Amazon

“We have used TLA+ on 10 large complex real-world systems. In every case TLA+ has added significant value, either finding subtle bugs that we are sure we would not have found by other means, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.”
## Use of TLA+ at Amazon

### Applying TLA+ to some of our more complex systems

<table>
<thead>
<tr>
<th>System</th>
<th>Components</th>
<th>Line count (excl. comments)</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Background redistribution of data</td>
<td>645 PlusCal</td>
<td>Found 1 bug, and found a bug in the first proposed fix.</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>Replication &amp; group-membership system</td>
<td>939 TLA+</td>
<td>Found 3 bugs, some requiring traces of 35 steps</td>
</tr>
<tr>
<td>EBS</td>
<td>Volume management</td>
<td>102 PlusCal</td>
<td>Found 3 bugs.</td>
</tr>
<tr>
<td>Internal distributed lock manager</td>
<td>Lock-free data structure</td>
<td>223 PlusCal</td>
<td>Improved confidence. Failed to find a liveness bug as we did not check liveness.</td>
</tr>
<tr>
<td></td>
<td>Fault tolerant replication and reconfiguration algorithm</td>
<td>318 TLA+</td>
<td>Found 1 bug. Verified an aggressive optimization.</td>
</tr>
</tbody>
</table>
First TLA+ Example
1-bit Clock

- Clock’s possible behaviours:

\[ b = 1 \rightarrow b = 0 \rightarrow b = 1 \rightarrow b = 0 \rightarrow \ldots \]

\[ b = 0 \rightarrow b = 1 \rightarrow b = 0 \rightarrow b = 1 \rightarrow \ldots \]
1-bit Clock

- State variable: $b$

- Initial predicate:
  $$b = 1 \lor b = 0$$

- Next-step action ($b'$ is the variable at the next state):
  $$\lor (b = 0) \land (b' = 1)$$
  $$\lor (b = 1) \land (b' = 0)$$

The initial state and next-step action are formulas in TLA
1-bit Clock

• State variable:
  \[ b \]

• Initial predicate:
  \[ b = 1 \lor b = 0 \]

• Next-step action (\( b' \) is the variable at the next state):
  \[
  \text{IF } b = 0 \text{ THEN } b' = 1 \\
  \text{ELSE } b' = 0 
  \]

The initial state and next-step action are formulas in TLA
1-bit Clock: TLA specification

----------------------------- MODULE OneBitClock -----------------------------
VARIABLE b

Init == (b=0) \lor (b=1)

Next == \lor b = 0 \land b' = 1
\lor b = 1 \land b' = 0

-----------------------------------------------------------------------------

What about the clock properties?
System’s properties

- **Safety**
  - Something bad never happens
  - E.g. system never deadlocks, the account balance is greater or equal to zero

- **Liveness**
  - Something good eventually happens
  - E.g. if a process request access to a critical region it will eventually be granted access, the light will eventually turn green

Let’s ignore liveness properties for now
1-bit Clock: TLA specification

---

**MODULE** OneBitClock

**VARIABLE** b

Init == (b=0) \(\lor\) (b=1)

TypeInv == b \(\in\) \{0,1\}

Next == \(\lor\) b = 0 \(\land\) b' = 1

\(\lor\) b = 1 \(\land\) b' = 0

Spec == Init \(\land\) \[\[]\text{Next}\_\ll<>b\gg\]

---

**THEOREM** Spec => \[\[]\text{TypeInv}\]
1-bit Clock: TLA specification

---

**MODULE** OneBitClock

**VARIABLE** b

Init == (b=0) ∨ (b=1)

TypeInv == b ∈ {0,1}

Next == ∨ b = 0 ∧ b' = 1
        ∨ b = 1 ∧ b' = 0

Spec == Init ∧ [][Next]_<<b>>

**THEOREM** Spec => []TypeInv

---

The initial state satisfies *Init*

Every transition satisfies *Next* or leaves *b* unchanged
1-bit Clock: TLA specification

MODULE OneBitClock

VARIABLE b

Init == (b=0) ∨ (b=1)

TypeInv == b \in {0,1}

Next == \(\lor b = 0 \land b' = 1\)
    \(\lor b = 1 \land b' = 0\)

Spec == Init ∩ [][Next]_<<b>>

THEOREM Spec => []TypeInv

Theorem specifies an invariant property
TLC model checker

• Exhaustive breadth-first search of all reachable states

• Finds (one of) the shortest path to the property violation
Computing all possible behaviours

- State graph is a directed graph $G$

1. Put into $G$ to the set of all initial states

2. For every state $s$ in $G$ compute all possible states $t$ such that $s \rightarrow t$ can be a step in a behaviour

3. For every state $t$ found in step 2 not in $G$, draw an edge from $s$ to $t$

4. Repeat the previous steps until no new states or edges can be added to $G$
TLC: state space progress

- Diameter
  - Number of states in the longest path of $G$ with no repeated states
- States found
  - Total number of states it examined in step 1 and 2
- Distinct states
  - Number of states that form the set of nodes of $G$
- Queue size
  - Number of states $s$ in $G$ for which step 2 has not yet been done
1-bit Clock: Model checking

- Checking the 1-bit clock with TLC model checker (demo)
Exercise 1

• Define a TLA+ specification of an hour clock

• Check with TLC the typing invariant
TLA+ Overview
EXTENDS M1,..., Mn
/* Incorporates the declarations, definitions, assumptions, and theorems from
* the modules named M1,...,Mn into the current module.

CONSTANTS C1,..., Cn /* Declares the C1,..., Cn to be constant parameters.

ASSUME P /* Asserts P as an assumption.

VARIABLES x1,..., xn /* Declares x1,..., xn as variables.

TypeInv == exp /* Declares the types of variables x1,..., xn.

Init == exp /* Initializes variables x1,..., xn.

F(x1,..., xn) == exp
/* Defines F to be an operator such that
* F(e1,...,en) equals exp with each identifier xk replaced by ek.

f[x \in S] == exp
/* Defines f to be the function with domain S such that f[x] = exp
* for all x in S.
* The symbol f may occur in exp, allowing a recursive definition.

THEOREM P
/* Asserts that P can be proved from the definitions and assumptions of the
* current module.
TLA+ syntax and semantics

• Logic
• Sets
• Functions
• Tuples, sequences and records
• EXCEPT, UNION, and CHOOSE operators
Logic

\( \land \quad \lor \quad \neg \quad \Rightarrow \quad \equiv \)

TRUE \quad FALSE \quad BOOLEAN \quad \{\text{the set \{TRUE, FALSE\}}\}

\( \forall x \in S : p \quad \exists x \in S : p \)

\(~(\text{TRUE} \land b)\)

\( a \Rightarrow b \)

Next == b' = 0

b \in BOOLEAN

x \notin S

\( \forall x \in \{1, 2, 3, 4, 5\} : x \geq 0 \)

\( \exists x \in \{1, 2, 3, 4, 5\} : x \% 2 = 0 \)
Sets

\[ S = \{1, 2, 3\} \]
\[ S \neq \{1, 2, 3\} \quad S \neq \{1, 2, 3\} \]
\[ x \in S \]
\[ x \notin S \]
\[ S \cup \{1, 2, 3\} \]
\[ \{ n \in \{1, 2, 3, 4, 5\} : n \% 2 \neq 0 \} = \{1, 3, 5\} \]
\[ \{ 2n+1 : n \in \{1, 2, 3, 4, 5\} \} = \{3, 5, 7, 9, 11\} \]
\[ \text{UNION} \{ \{1, 2\}, \{2, 3\}, \{3, 4\} \} = \{1, 2, 3, 4\} \]
\[ \text{SUBSET} \{1, 2\} = \{\}, \{1\}, \{2\}, \{1, 2\}\]
CHOOSE $x \in S : P(x)$

/* Equals some value $v$ in $S$ such that $P(v)$ equals true, if such a value exists. */
/* Its value is unspecified if no such $v$ exists */

CHOOSE $x \in \{1, 2, 3, 4, 5\} : \text{TRUE}$

CHOOSE $x \in \{1, 2, 3, 4, 5\} : x \% 2 = 0$

CHOOSE is deterministic!
CHOICE vs. non-determinism

removeOneDet ==
IF procs \= {}
THEN procs' =
    procs \ {CHOOSE t \in procs : TRUE}
ELSE UNCHANGED procs

removeOneNonDet ==
IF procs \= {}
THEN \E x \in procs : procs' = procs \ {x}
ELSE UNCHANGED waiting

Deterministic

a single successor state

Non-deterministic

many of successor states
Functions

\[
\begin{align*}
  f[e] & \quad \text{[Function application]} \\
  \text{DOMAIN } f & \quad \text{[Domain of function } f\text{]} \\
  [x \in S \mapsto e] & \quad \text{[Function } f \text{ such that } f[x] = e \text{ for } x \in S\text{]} \\
  [S \to T] & \quad \text{[Set of functions } f \text{ with } f[x] \in T \text{ for } x \in S\text{]} \\
  [f \text{ EXCEPT } ![e_1] = e_2] & \quad \text{[Function } \tilde{f} \text{ equal to } f \text{ except } \tilde{f}[e_1] = e_2. \text{ An @ in } e_2 \text{ equals } f[e_1].]\end{align*}
\]

\[
\begin{align*}
  [i \in \{2,3,5,9\} \mapsto i - 7] & = (2 :> -5 \ @@ \ 3 :> -4 \ @@ \ 5 :> -2 \ @@ \ 9 :> 2) \\
  \text{DOMAIN} \ [i \in \{2,3,5,9\} \mapsto i - 7] & = \{2, 3, 5, 9\} \\
  [ [i \in \{2,3,5,9\} \mapsto i - 7][3] & = -4 \\
  [ \{2,4\} \mapsto \{"a", \ "b" \} ] & = \{ (2 \mapsto "a" \ @@ \ 4 \mapsto "a"), (2 \mapsto "a" \ @@ \ 4 \mapsto "b"), \\
  & \quad (2 \mapsto "b" \ @@ \ 4 \mapsto "a"), (2 \mapsto "b" \ @@ \ 4 \mapsto "b") \} \\
  [ [i \in \{2,3,5,9\} \mapsto i - 7] \text{ EXCEPT } ![2]= 12 \ ] & = \\
  & \quad (2 \mapsto 12 \ @@ \ 3 \mapsto -4 \ @@ \ 5 \mapsto -2 \ @@ \ 9 :> \ 2) \end{align*}
\]
Records

\[ e.h \]  
\[ [h_1 \mapsto e_1, \ldots, h_n \mapsto e_n] \]  
\[ [h_1 : S_1, \ldots, h_n : S_n] \]  
\[ [r \text{ EXCEPT } !.h = e] \]  

[The \( h \)-component of record \( e \)]  
[The record whose \( h_i \) component is \( e_i \)]  
[Set of all records with \( h_i \) component in \( S_i \)]  
[Record \( \hat{r} \) equal to \( r \) except \( \hat{r}.h = e \). An @ in \( e \) equals \( r.h \).]

[node \( \mapsto "n1" \), edge \( \mapsto "e1" \)]  
[node \( \mapsto "n1" \), edge \( \mapsto "e1" \)] \[.edge = "e1" \]  
[nodes : \{"n1","n2"\}, edges : \{"e1","e2"\}]  
[node \( \mapsto "n1" \), edge \( \mapsto "e1" \)] \[ \text{EXCEPT } !.edge = "xpt0" \] =  
[node \( \mapsto "n1" \), edge \( \mapsto "xpt0" \)]
Tuples

\[ e[i] \quad \text{[The } i^{\text{th}} \text{ component of tuple } e] \]
\[ \langle e_1, \ldots, e_n \rangle \quad \text{[The } n\text{-tuple whose } i^{\text{th}} \text{ component is } e_i] \]
\[ S_1 \times \ldots \times S_n \quad \text{[The set of all } n\text{-tuples with } i^{\text{th}} \text{ component in } S_i] \]

<<"ana", 32, 37495>>

<<"ana", 32>>[2] = 32

<<"ana", 32>>[1] = "ana"

\{1,2,3\} \times \{"a","b"\} = \{ <<1, "a">>, <<1, "b">>, <<1, "c">>, <<2, "a">>, <<2, "b">>, <<2, "c">>, <<3, "a">>, <<3, "b">>, <<3, "c">> \}
Sequences

MODULE Sequences

LOCAL INSTANCE Naturals

Seq(S) == UNION {[1..n -> S] : n \in Nat}

Len(s) == CHOOSE n \in Nat : DOMAIN s = 1..n

s \o t == [i \in 1..(Len(s) + Len(t)) |-> IF i \leq Len(s) THEN s[i] ELSE t[i-Len(s)]]

Append(s, e) == s \o <<e>>

Head(s) == s[1]

Tail(s) == [i \in 1..(Len(s)-1) |-> s[i+1]]

SubSeq(s, m, n) == [i \in 1..(1+n-m) |-> s[i+m-1]]
Other constructs

\[
\begin{align*}
\text{IF } p \text{ THEN } e_1 \text{ ELSE } e_2 & \quad [e_1 \text{ if } p \text{ true, else } e_2] \\
\text{CASE } p_1 \rightarrow e_1 \square \ldots \square p_n \rightarrow e_n & \quad [\text{Some } e_i \text{ such that } p_i \text{ true}] \\
\text{CASE } p_1 \rightarrow e_1 \square \ldots \square p_n \rightarrow e_n \square \text{OTHER} \rightarrow e & \quad [\text{Some } e_i \text{ such that } p_i \text{ true, or } e \text{ if all } p_i \text{ are false}] \\
\text{LET } d_1 \triangleq e_1 \ldots d_n \triangleq e_n \text{ IN } e & \quad [e \text{ in the context of the definitions}] \\
\wedge p_1 \text{ [the conjunction } p_1 \wedge \ldots \wedge p_n] & \quad \lor p_1 \text{ [the disjunction } p_1 \lor \ldots \lor p_n] \\
\vdots & \vdots \\
\wedge p_n & \lor p_n
\end{align*}
\]
A farmer is on one shore of a river and has with him a fox, a chicken, and a sack of grain.

He has a boat that fits one item besides himself.

In the presence of the farmer nothing gets eaten, but if left without the farmer, the fox will eat the chicken, and the chicken will eat the grain.

How can the farmer get all three items across the river safely?
Exercise: Crossing the river

• Define a TLA+ specification for this problem.

• Check with TLC the typing invariant.

• Add an invariant stating that it is not possible to get all three items across the river.

• Use TLC to find a solution to this problem.
Exercise: Crossing the river

• Some help:

```
----------------------------- MODULE CrossingRiver -----------------------------
EXTENDS Integers

CONSTANTS Farmer, Fox, Chicken, Grain

Items == {Fox, Chicken, Grain}

safe(S) == ~(Fox, Chicken) \subseteq S \lor (Chicken, Grain) \subseteq S

VARIABLES onLeftShore, onRightShore

TypeInv ==
    \forall onLeftShore \in SUBSET (Items \union \{Farmer\})
    \forall onRightShore \in SUBSET (Items \union \{Farmer\})
```
Crossing the river: Solution

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crossWithItem {Farmer,Chicken}</td>
<td>State (num = 1) {Grain, Chicken, Farmer, Fox} {}</td>
</tr>
<tr>
<td>crossAlone {Farmer}</td>
<td>State (num = 2) {Grain, Fox} {Chicken, Farmer}</td>
</tr>
<tr>
<td>crossWithItem {Farmer,Grain}</td>
<td>State (num = 3) {Grain, Farmer, Fox} {Chicken}</td>
</tr>
<tr>
<td>crossWithItem {Farmer,Chicken}</td>
<td>State (num = 4) {Fox} {Grain, Chicken, Farmer}</td>
</tr>
<tr>
<td>crossWithItem {Farmer,Fox}</td>
<td>State (num = 5) {Chicken, Farmer, Fox} {Grain}</td>
</tr>
<tr>
<td>crossWithItem {Farmer,Chicken}</td>
<td>State (num = 6) {Chicken} {Grain, Farmer, Fox}</td>
</tr>
<tr>
<td>crossAlone {Farmer}</td>
<td>State (num = 7) {Chicken, Farmer} {Grain, Fox}</td>
</tr>
<tr>
<td>crossWithItem {Farmer,Chicken}</td>
<td>State (num = 8) {} {Grain, Chicken, Farmer, Fox}</td>
</tr>
</tbody>
</table>